Aurora 参考文档

Version: 1.0

目录

前言

xxxxx

1. 文档说明

各个章节相关负责人以及审校人员

表 1. Aurora 开发团队

序号	标题	撰写	Email
#1	APL总体架构概述	牛佳庆	njq.niu@hand-china.com
#2	DataSet	牛佳庆	njq.niu@hand-china.com
#3	界面布局	牛佳庆	njq.niu@hand-china.com
#4	Grid组件	牛佳庆	njq.niu@hand-china.com
#5	编辑组件	吴华真	huazhen.wu@hand-china.com

关于我们

Aurora 开发团队
隶属于上海汉得信息技术股份有限公司(www.hand-china.com)MAS部门.

2. 版权声明

第 1 章 Aurora 框架简介

1.1. 前言

Aurora 是一个开源的web框架。

第 2 章 Aurora开发概览

2.1. 前言

本章是面向Aurora初学者的一个入门教程。

本章面向Aurora初学者，但需要XML,JavaScript和SQL知识。

2.2. 安装配置

安装配置。

2.2.1. 下载框架程序

下载框架程序

2.2.2. 配置

配置文件

2.3. 基本的数据维护表单

基本的数据维护表单

2.4. 数据查询及展示界面

数据查询及展示界面。

2.5. 完整的数据维护界面

完整的数据维护界面。

2.6. 主从级联界面

主从级联界面。

2.7. 应用Feature

主从级联界面。

2.7.1. 多语言

多语言。

2.7.2. 系统编码

系统编码。

2.8. 总结

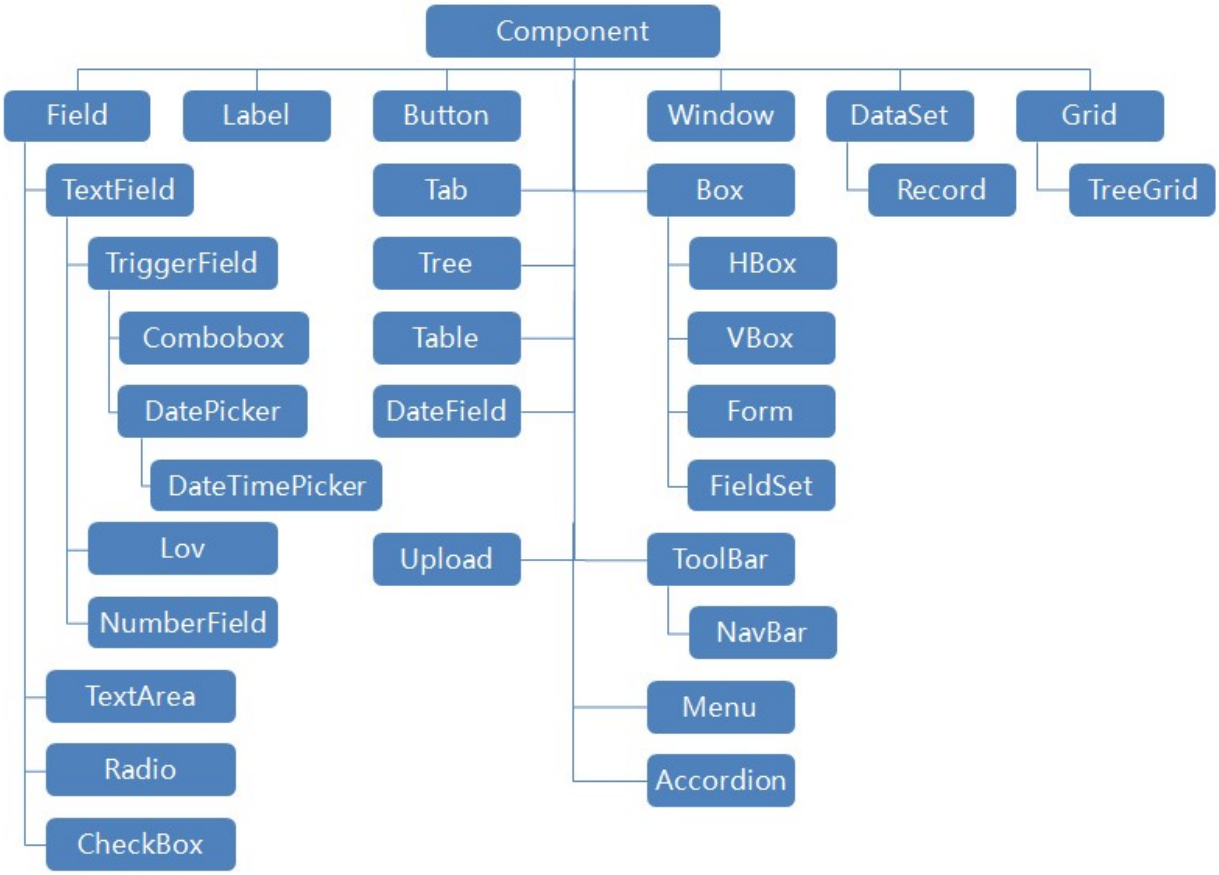
总结。

第 3 章 Aurora展示层(APL)

3.1. APL(Aurora Presentation Layer)概述

APL提供了一整套跨浏览器的UI组件库.所有的UI组件都是基于JavaScript和Html构建的,运行在客户端浏览器中.用户可以在客户端通过调用相应的函数改变UI组件的状态和行为.

APL UI组件结构图:



上图展示了所有APL UI组件的继承关系.

3.2. APL服务端工作原理

APL服务端简介

3.2.1. Aurora如何创建用户界面

正文...

3.2.2. 数据容器

正文...

3.2.3. 组件映射

正文...

3.2.4. BuildSession

正文...

3.2.5. 事件机制

正文...

3.2.6. 资源文件处理

正文...

3.2.7. 动态组件配置

正文...

3.3. APL UI组件

APL 提供了各种丰富的UI组件，例如布局组件(HBox,VBox,Form..)，数据展现组件(Grid,Table,TreeGrid等),数据容器组件(DataSet等).

APL UI组件兼容以下类型的浏览器

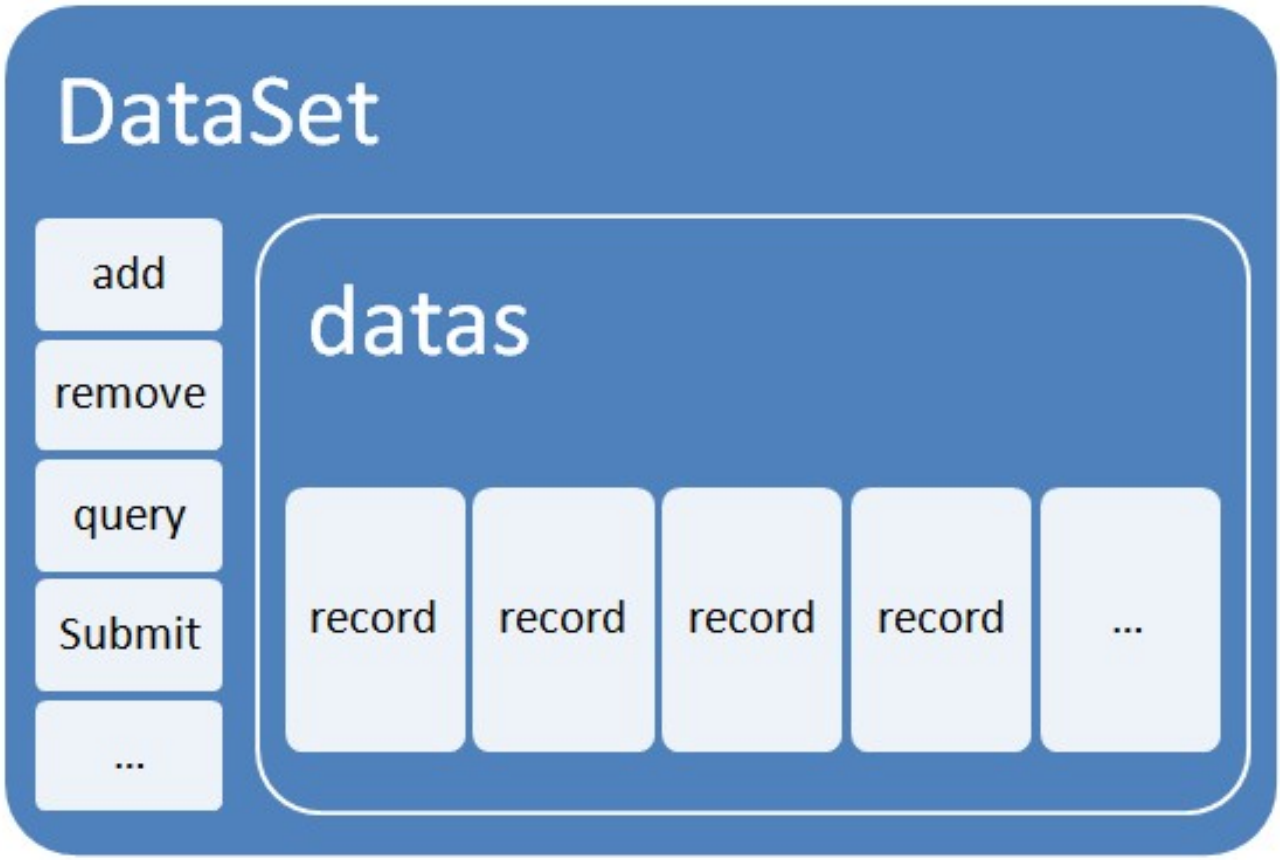
3.3.1. 核心Javascript API

APL采用了第三方的开源库Ext Core 3.0 (http://www.sencha.com/products/extcore/)作为底层的开发库,在此基础上构建了丰富的组件库.

3.3.2. DataSet

什么是DataSet?

DataSet是一个运行在客户端浏览器中的组件,本质上来说是一个JavaScript构建的对象. DataSet是一个数据容器,他封装了常用的一些数据操作,我们可以通过下面这张图来更深刻的理解DataSet的含义



通过上图我们可以看到DataSet是一个数据容器,它包含了一个数组对象用来存放所有的record对象.

Record代表一条数据对象,DataSet和Record的关系我们可以这样理解:假如我们把dataset比作数据库中的一张表, 那么record就是表中的一行记录.

3.3.2.1. DataSet定义

在screen文件中我们通过<a:dataSet>标签来定义一个dataset对象

```
<a:dataSet model="sys.sys_user" id="sys_user_create_ds">
    <a:fields>
        <a:field name="user_name" required="true"/>
        <a:field name="start_date" required="true" validator="dateValidator"/>
        <a:field name="description" required="true"/>
    </a:fields>
    <events>
        <a:event name="submitsuccess" handler="onCreateUserSuccess"/>
        <a:event name="update" handler="onUpdate"/>
    </events>
</a:dataSet>
```

每一个dataset都应该定义一个id属性,在整个screen文件中不得出现重复的id值. 定义了id值我们可以在页面脚本中通过$('sys_user_create_ds')的方式 获取到这个dataset对象,进而可以调用相应的函数方法.

fields子节点定义了这个dataset都包含哪些field以及field中的特性. 在field上我们要指定它的name,通过ajax获取到的json数据会根据name来匹配. 在field上我们还可以定义一些其他的附加特性,例如是否必输,是否只读等等.

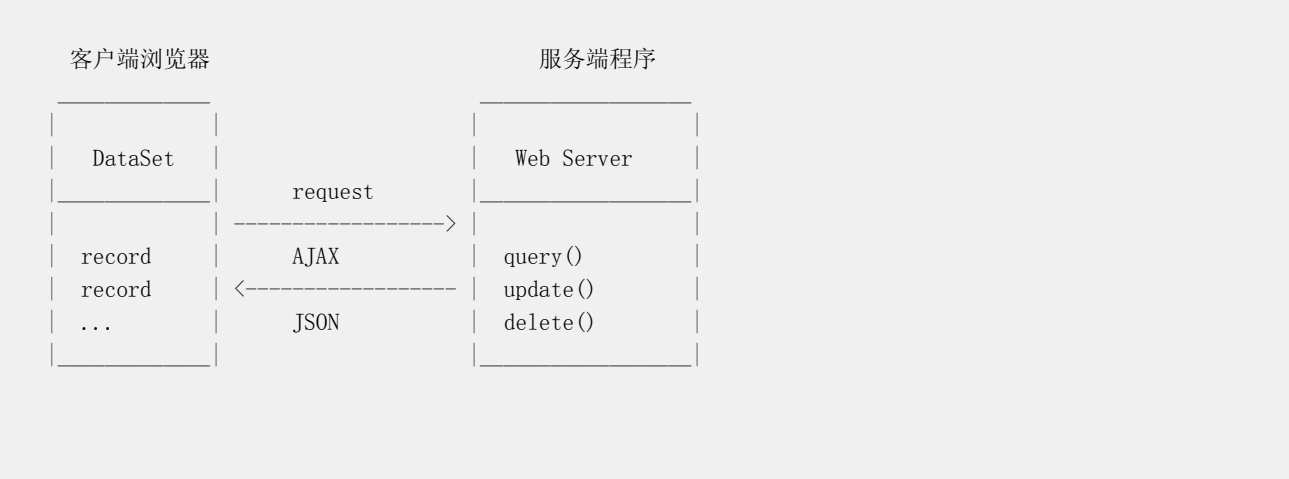
events标签定义了需要响应的事件函数,例如update事件,当dataset对其中一条record的field进行更新的时候,dataset会触发一个update事件, 这样我们可以通过配置一个客户端函数onUpdate来响应.

```
function onUpdate(ds,record,name,value){
    if(name == 'user_password' || name == 'user_password_re'){
        record.validate((name == 'user_password') ? 'user_password_re' : 'user_password');
    }
    if(name == 'start_date' || name == 'end_date'){
        record.validate((name == 'start_date') ? 'end_date' : 'start_date');
    }
}
```

更多详细配置可以参考AuroraTagDocument文档

3.3.2.2. 前言

DataSet是一个客户端的JavaScript组件,它封装了相应的ajax操作,用来和服务端进行数据交互.



DataSet是一个客户端的JavaScript组件,它封装了相应的ajax操作,用来和服务端进行数据交互.

DataSet提供了基本的数据操作,主要分为两类.一类是客户端操作,一类是和服务端通信.

- 客户端操作 -- 例如当调用add函数后,其实本质上是在客户端dataset中增加一条record记录,这个并没有同步到服务端数据库中.
- AJAX操作 -- 通过AJAX调用和服务端进行通讯,例如query查询服务端返回相应的json数据,填充到客户端.

3.3.2.3. 元数据(Metadata)

Metadata元数据主要用来描述field的附加特性,例如是否只读,是否必输等等.

例如:设置某一字段必输,我们可以在配置dataset的时候指定对应field的required值

```
<a:dataSet id="exp_employee_group_result_ds">
    <a:fields>
        <a:field name="expense_user_group_code" required="true"/>
```

```
        <a:field name="description" required="true"/>
    </a:fields>
</a:dataSet>
```

例如:设置某一字段只读,我们可以在配置dataset的时候指定对应field的readOnly值

```
<a:dataSet id="exp_employee_group_result_ds">
    <a:fields>
        <a:field name="expense_user_group_code" readOnly="true"/>
        <a:field name="description" required="true"/>
    </a:fields>
</a:dataSet>
```

3.3.2.4. 校验(Validate)

很多情况下我们需要对dataset的值进行校验,这个时候我们可以通过在field上配置校验函数(validator)来实现

例如:我们对2个日期字段进行校验,规则是结束日期不得小于开始日期.首先我们在dataset的2个日期field上配置validator

```
<a:dataSet id="fnd_companies_create_ds"  model="fnd.fnd_companies">
    <a:fields>
        ...
        <a:field name="start_date_active" datatype="date" required="true" validator="dateValidator"/>
        <a:field name="end_date_active" datatype="date" validator="dateValidator"/>
    </a:fields>
</a:dataSet>
```

接下来我们需要实现校验函数dateValidator

```
function dateValidator(record, name, value){
    if(name == 'start_date_active' || name == 'end_date_active'){
        var start_date = record.get('start_date_active');
        var end_date = record.get('end_date_active');
        if(typeof(end_date) != 'undefined' && !Ext.isEmpty(end_date)){
            if(!compareDate(start_date, end_date)){
                return '开始时间不能大于结束时间';
            }
        }
        return true;
    }
}
```

2个日期field公用了同一个校验函数,所以首先要判断name值,然后分别通过record获取对应的开始和结束日期. 如果校验成功返回true,校验失败返回提示信息.

3.3.2.5. 动态界面逻辑处理

待更新...

3.3.2.6. DataSet常用操作与事件

DataSet 常用函数

表 3.1. DataSet相关函数

函数	说明
add	在客户端dataset中新增一条record记录
remove	在客户端dataset中删除指定的record
query	通过指定的url查询数据,服务端返回json数据填充到客户端dataset中
submit	降dataset中的数据提交到指定的url中

DataSet 常用事件

表 3.2. DataSet事件

事件名	说明
add	新增一条record后触发
remove	删除record后触发
update	当dataset中的record被更新后触发
load	当dataset成功加载数据后触发
submit	当dataset提交请求时触发

更多详细的函数请参考AuroraJavaScriptDocument

3.3.3. 界面布局(Layout)

APL的布局是基于服务器端生成,这和其他的基于客户端布局的开源框架有点不同.APL的布局基本上是通过table在服务端事先生成好的,这样的好处在于可以 减少客户端机器的压力,充分利用服务器的资源优势.

APL的布局主要是由Box,VBox,HBox,Form,FieldSet等容器组件组成.

3.3.3.1. Box

Box组件类似html中的table标签,通过行(row)和列(column)的配置来构建一个表格.

在screen文件中我们通过<a:box>标签来定义一个box对象,然后定义row和column.

box标签下的组件将会按照从左到右从上到下的原则进行排列.

```
<a:box column="2" row="3">
    <a:textField name="cmp1"/>
    <a:textField name="cmp2"/>
    <a:textField name="cmp3"/>
    <a:textField name="cmp4"/>
    <a:textField name="cmp5"/>
    <a:textField name="cmp6"/>
</a:dataSet>
```

布局实例图:

3.3.3.2. VBox

VBox组件继承自Box组件,VBox的column恒等于1,所以VBox相当于column=1的Box组件.

VBox下的组件按照自上而下的方式进行布局

```
<a:vBox>
    <a:textField name="cmp1"/>
    <a:textField name="cmp2"/>
    <a:textField name="cmp3"/>
</a:vBox>
```

布局实例图:

3.3.3.3. HBox

HBox组件继承自Box组件,HBox的row恒等于1,所以HBox相当于row=1的Box组件.

HBox下的组件按照从左到右的方式进行布局

```
<a:hBox>
    <a:textField name="cmp1"/>
    <a:textField name="cmp2"/>
    <a:textField name="cmp3"/>
</a:hBox>
```

布局实例图:

```
    HBox
 ____________________________________
|            |            |          |
|    cmp1    |    cmp2    |    cmp3  |
|____________|____________|__________|
```

3.3.3.4. Form

Form组件继承自Box组件,带有一个title头.

注意:Form和html中的form标签是有本质区别的.Form组件仅仅是一个布局容器,并没有提交数据的功能.

Form组件的布局方式和Box类似,根据row和column的布局所有组件,采取从上到下从左到右的方式.

```
<a:form id="loginForm" labelWidth="100" row="1" title="Form Title" width="320">
    <a:textField name="cmp1"/>
    <a:textField name="cmp2"/>
    <a:textField name="cmp2"/>
</a:form>
```

布局实例图:

```
 Form
 ____________________________________
|  Form Title                        |
|____________________________________|
|            |            |          |
|    cmp1    |    cmp2    |    cmp3  |
|____________|____________|__________|
```

提示:Form和Box以及其他容器组件类似,都是可以嵌套的.举例来说Form下可以再次嵌套VBox,HBox,Form标签.

3.3.3.5. FieldSet

FieldSet组件继承自Box组件,带有一个title头. 它和Form标签的不同仅仅局限在界面展现上.

提示:FieldSet的配置和Form基本类似,它和Form标签的不同仅仅局限在界面展现上.

FieldSet组件的布局方式和Box类似,根据row和column的布局所有组件,采取从上到下从左到右的方式.

```
<a:fieldSet id="loginForm" labelWidth="100" row="1" title="Form Title" width="320">
    <a:textField name="cmp1"/>
    <a:textField name="cmp2"/>
    <a:textField name="cmp2"/>
</a:fieldSet>
```

布局实例图:

```
    FieldSet

   |---Form Title---------------------------|
   |________________________________________|
   |              |              |          |
   |    cmp1      |    cmp2      |    cmp3  |
   |______________|______________|__________|
```

提示:FieldSet和Box以及其他容器组件类似,都是可以嵌套的.举例来说FieldSet下可以再次嵌套VBox,HBox,Form标签.

3.3.4. 编辑组件

编辑组件是提供文本输入编辑及选择功能的组件，主要是由TextField,NumberField,ComboBox,DatePicker,Lov,DateTimePicker,TextArea,Radio,CheckBox等组件组成。

编辑组件也可作为Grid组件的Editor，具体请参阅Grid的编辑器。

3.3.4.1. TextField

TextField是一个提供文本输入编辑的组件，可限制大小写的输入。

上图是TextField组件在页面中的呈现，输入框前的文字信息是通过TextField标签属性prompt来定义的。

3.3.4.1.1. TextField定义

在screen文件中我们通过<a:textField>标签来定义一个TextField对象。

```
```

```
<a:textField bindTarget="login_dataset" id="user_name_tf" name="user_name"
        prompt="HAP_USERNAME" width="150" typeCase="upper">
    <a:events>
        <a:event handler="login" name="enterdown"/>
    </a:events>
</a:textField>
```

textField标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

textField标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将TextField对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在TextField上，另外TextField上的文字编辑也会立刻修改dataset中的数据。

events标签定义了需要响应的事件函数，例如enterdown事件，当键盘键入回车键时，TextField会触发enterdown事件，这样我们可以通过配置一个客户端函数login来响应。

```
function login(){
    var lds = $('login_dataset');
    var record = lds.getCurrentRecord();
    Aurora.request({url:'login.svc', para:record.data, success:function(){
        window.location.href='role_select.screen'
    },scope:this});
}
```

3.3.4.1.2. TextField标签属性

表 3.3.

属性名	用途	默认值	是否必填
bindTarget	组件所绑定的dataset数据集，属性值是dataset的ID。		
className	组件的样式表。		
emptyText	当组件没有值的时候显示在组件上的提示信息。		
id	组件的唯一标识，可用$(id)方法获得组件对象。		
marginWidth	组件与窗口之间的宽度差，单位像素(px)，可以根据窗口宽度的改变而改变。		
name	组件对应dataset数据集中的一个字段field，属性值是字段名。		
prompt	输入框前的提示文字，默认调用BM的prompt。		
readOnly	设定组件是否只读。 取值 true | false	false	
required	设定组件是否必填。	false	

属性名	用途	默认值	是否必填
	取值 true | false		
style	组件的样式。		
typeCase	组件的大小写输入限制。 取值 upper | lower		
width	组件的宽度，单位像素(px)。	150	

3.3.4.1.3. TextField对象事件

表 3.4.

事件名	用途
blur	失去焦点时触发的事件。
change	文本内容发生改变时触发的事件。
enterdown	敲击回车键时触发的事件。
focus	获得焦点时触发的事件。
keydown	键盘按下时触发的事件。
keypress	键盘敲击时触发的事件。
keyup	键盘抬起时触发的事件。
mouseover	鼠标移到组件上时触发的事件。
mouseout	鼠标移出组件时触发的事件。

3.3.4.2. NumberField

NumberField是一个提供数字输入编辑的组件，继承自TextField组件，拥有TextField标签的属性以及TextField对象的方法和事件。

上图是NumberField组件在页面中的呈现，输入框前的文字信息是通过NumberField标签属性prompt来定义的。

3.3.4.2.1. NumberField定义

在screen文件中我们通过<a:numberField>标签来定义一个NumberField对象。

```
<a:numberField bindTarget="fnd_tax_type_codes_query_ds" name="tax_type_rate"
        allowDecimals="true" allowFormat="true" decimalPrecision="1">
    <a:events>
```

```
        <a:event name="enterdown" handler="queryTaxTypeCodes"/>
    </a:events>
</a:numberField>
```

numberField标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

numberField标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将NumberField对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在NumberField上，另外NumberField上的文字编辑也会立刻修改dataset中的数据。

events标签定义了需要响应的事件函数，例如enterdown事件，当键盘键入回车键时，NumberField会触发enterdown事件，这样我们可以通过配置一个客户端函数queryTaxTypeCodes来响应。

```
function queryTaxTypeCodes(){
    $('fnd_tax_type_codes_result_ds').query();
}
```

3.3.4.2.2. NumberField标签属性

表 3.5.

属性名	用途	默认值	是否必填
allowDecimals	是否允许NumberField的值为小数。 取值 true | false	true	
allowNegative	是否允许NumberField的值为负数。 取值 true | false	true	
allowFormat	是否允许NumberField的值按照千分位显示。 取值 true | false	true	
decimalPrecision	组件的小数位精度，必须在allowDecimals为true的情况下才能使用。	2	
其他	参阅TextField标签属性。		

3.3.4.2.3. NumberField对象事件

请参阅TextField的对象事件。

3.3.4.3. ComboBox

ComboBox是一个可输入的下拉框组件，继承自TextField组件，拥有TextField标签的属性以及TextField对象的方法和事件。其主要的功能就是运用键值对的形式将数据(键)以值的形式来呈现。

上图是ComboBox组件在页面中的呈现，输入框前的文字信息是通过ComboBox标签属性prompt来定义的。

3.3.4.3.1. ComboBox定义

在screen文件中我们通过<a:comboBox>标签来定义一个ComboBox对象。

```
<a:dataSet id="sys_user_islocked_ds">
    <a:datas>
        <a:record name="已冻结" code="Y"/>
        <a:record name="未冻结" code="N"/>
    </a:datas>
</a:dataSet>
<a:dataSet id="sys_user_query_ds">
    <a:fields>
        <a:field name="user_name"/>
        <a:field name="frozen_flag_display" displayField="name" options="sys_user_islocked_ds"
                returnField="frozen_flag" valueField="code"/>
    </a:fields>
</a:dataSet>
<a:comboBox name="frozen_flag_display" bindTarget="sys_user_query_ds" prompt="SYS_USER.IS_FROZEN">
    <a:events>
        <a:event handler="login" name="enterdown"/>
    </a:events>
</a:comboBox>
```

comboBox标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

comboBox标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将ComboBox对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在ComboBox上，另外ComboBox上的文字编辑也会立刻修改dataset中的数据。

comboBox所绑定的dataset应该至少有两条字段(field)，一条是对应数据库的数据的隐式字段，另一条是用来显示文字的显示字段。如上代码所示，comboBox的选项是在显示字段(如上代码中name="frozen_flag_display"的field)标签上加上returnField,options,displayField,valueField属性组合实现的。其中returnField属性是隐式字段名，表示comboBox选中选项后用哪个字段来承载这个值，进而通过dataset的方法用来对数据库进行添加修改等操作。options属性是一个选项的数据集的ID，该数据集的所有记录(record)都应该有两条字段(field)，一条字段是选项的值(如上代码中code="Y"的code)，另一条字段是选项显示的文本(如上代码中name="已冻结"的name)，valueField和displayField属性分别指定的就是这两条字段的名称(code和name)。

events标签定义了需要响应的事件函数，例如enterdown事件，当键盘键入回车键时，ComboBox会触发enterdown事件，这样我们可以通过配置一个客户端函数login来响应。

```
function login(){
```

```
    var lds = $('login_dataset');
    var record = lds.getCurrentRecord();
    Aurora.request({url:'login.svc', para:record.data, success:function(){
        window.location.href='role_select.screen'
    },scope:this});
}
```

3.3.4.3.2. ComboBox标签属性

请参阅TextField的标签属性。

3.3.4.3.3. ComboBox对象事件

表 3.6.

事件名	用途
select	选择选项时触发的事件。
其他	请参阅TextField对象的事件

3.3.4.4. DateField

DateField正文 ...

3.3.4.5. DatePicker

DatePicker是一个提供日期输入编辑的组件，继承自TextField组件，拥有TextField标签的属性以及TextField对象的方法和事件。

上图是DatePicker组件在页面中的呈现，输入框前的文字信息是通过DatePicker标签属性prompt来定义的。

3.3.4.5.1. DatePicker定义

在screen文件中我们通过<a:datePicker>标签来定义一个DatePicker对象。

```
<a:datePicker name="start_date" bindTarget="sys_user_create_ds" viewSize="2"
        enableBesideDays="both" enableMonthBtn="both">
    <a:events>
        <a:event handler="login" name="enterdown"/>
    </a:events>
</a:datePicker>
```

datePicker标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

datePicker标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将DatePicker对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在DatePicker上，另外DatePicker上的文字编辑也会立刻修改dataset中的数据。

events标签定义了需要响应的事件函数，例如enterdown事件，当键盘键入回车键时，DatePicker会触发enterdown事件，这样我们可以通过配置一个客户端函数login来响应。

```
function login(){
    var lds = $('login_dataset');
    var record = lds.getCurrentRecord();
    Aurora.request({url:'login.svc', para:record.data, success:function(){
        window.location.href='role_select.screen'
    },scope:this});
}
```

3.3.4.5.2. DatePicker标签属性

表 3.7.

属性名	用途	默认值	是否必填
dayRenderer	日期渲染函数。dayRenderer属性指定一个回调函数的函数名，该函数可带三个参数，依次为cell,date,text。 cell - 显示日期的单元格，当cell.disabled=true时，该日期无法被选择。 date - 日期对应的date对象。 text - 日期所显示的文本，函数的反回值需为包含此参数的HTML字符串。		
enableBesideDays	enableBesideDays属性指定当月日期表是否显示上月结尾和(或)下月开头的日期。 取值 both | none | pre | next	both	
enableMonthBtn	enableMonthBtn属性指定日期表是否显示上月按钮和(或)下月按钮。	both	

属性名	用途	默认值	是否必填
	取值 both | none | pre | next		
viewSize	显示日期表的个数，已当前月开始依次排列。最大值是4。	1	

3.3.4.5.3. DatePicker对象事件

表 3.8.

事件名	用途
select	选择日期时触发的事件。
其他	请参阅TextField对象的事件

3.3.4.6. Lov

Lov是一个提供文本输入编辑和通过弹出窗口提供选项选择的组件，继承自TextField组件，拥有TextField标签的属性以及TextField对象的方法和事件。弹出窗口中为一个固定格式的页面，包含选项集的查询条件输入框和用Grid组件成列的选项集。

上图是Lov组件在页面中的呈现，输入框前的文字信息是通过Lov标签属性prompt来定义的。

3.3.4.6.1. Lov定义

在screen文件中我们通过<a:lov>标签来定义一个Lov对象。

```
<a:dataSet id="gld_exchange_rate_ds" autocreate="true">
    <a:datas dataSource="/model/gerc"/>
    <a:fields>
            <a:field name="currency_code_frn" lovGridHeight="300" lovHeight="460"
                lovService="gld.gld_currency_lov?currency_code_frn=${/model/gerc/record/@currency_code}"
                lovWidth="490" title="币种选择">
            <mapping>
                <map from="currency_code" to="currency_code_frn"/>
                <map from="currency_name" to="currency_name_frn"/>
            </mapping>
        </a:field>
    </a:fields>
    <a:field name="currency_name_frn" readonly="true"/>
</a:dataSet>

<a:lov name="currency_code_frn" bindTarget="gld_exchange_rate_ds" prompt="GLD_CURRENCY.CURRENCY_CODE_FRN">
    <a:events>
        <a:event handler="login" name="enterdown"/>
    </a:events>
</a:lov>
```

lov标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

lov标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将Lov对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在Lov上，另外Lov上的文字编辑也会立刻修改dataset中的数据。

如上代码，field标签是lov所绑定的dataset中的一条字段(field)，field标签中lovHeight,lovWidth属性可指定弹出窗口的高度和宽度，lovService属性可指定弹出窗口中生成页面的BM，title属性可指定弹出框的标题。注意：以上属性均可定义在lov标签上，效果相同，但是建议定义在field标签上。

field标签下的mapping标签定义了弹出窗口选项集的字段和主窗口的字段的联系关系，每条map标签都有from和to属性，from是lov弹出窗口中选项的字段，to是lov在主窗口绑定的dataset中的字段。

events标签定义了需要响应的事件函数，例如enterdown事件，当键盘键入回车键时，TextField会触发enterdown事件，这样我们可以通过配置一个客户端函数login来响应。

```
function login(){
    var lds = $('login_dataset');
    var record = lds.getCurrentRecord();
    Aurora.request({url:'login.svc', para:record.data, success:function(){
        window.location.href='role_select.screen'
    },scope:this});
}
```

3.3.4.6.2. Lov标签属性

表 3.9.

属性名	用途	默认值	是否必填
lovGridHeight	弹出窗口中Grid的高度。	350	
lovHeight	弹出窗口的高度。	400	
lovService	弹出窗口中数据对应的BM。		
lovUrl	弹出窗口中页面的Url。		
lovWidth	弹出窗口的宽度。	400	
title	弹出窗口的标题。		
其他	请参阅TextField的标签属性。		

3.3.4.6.3. TextField对象事件

表 3.10.

事件名	用途
commit	文本输入框中文本发生改变后触发校验查询之后或者将弹出窗口的选项被选择后触发的事件。
其他	请参阅TextField的对象事件

3.3.4.7. DateTimePicker

DateTimePicker是一个提供日期及时间输入编辑的组件，继承自DatePicker组件，拥有DatePicker标签的属性以及DatePicker对象的方法和事件。

上图是DateTimePicker组件在页面中的呈现，输入框前的文字信息是通过DateTimePicker标签属性prompt来定义的。

3.3.4.7.1. DateTimePicker定义

在screen文件中我们通过<a:dateTimePicker>标签来定义一个DateTimePicker对象。

```
<a:dateTimePicker name="start_date" bindTarget="sys_user_create_ds" viewSize="2"
        enableBesideDays="both" enableMonthBtn="both">
    <a:events>
        <a:event handler="login" name="enterdown"/>
    </a:events>
</a:dateTimePicker>
```

dateTimePicker标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

dateTimePicker标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将DateTimePicker对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在DateTimePicker上，另外DateTimePicker上的文字编辑也会立刻修改dataset中的数据。

events标签定义了需要响应的事件函数，例如enterdown事件，当键盘键入回车键时，DateTimePicker会触发enterdown事件，这样我们可以通过配置一个客户端函数login来响应。

```
function login(){
    var lds = $('login_dataset');
    var record = lds.getCurrentRecord();
    Aurora.request({url:'login.svc', para:record.data, success:function(){
        window.location.href='role_select.screen'
    },scope:this});
}
```

3.3.4.7.2. DateTimePicker标签属性

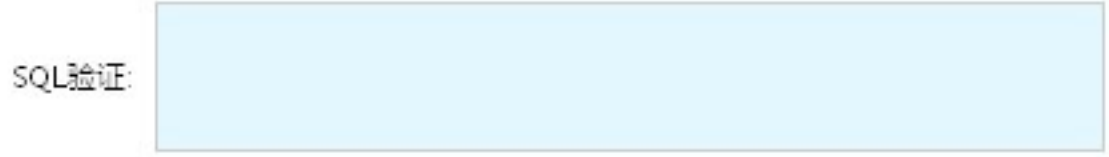
请参阅DatePicker的标签属性。

3.3.4.7.3. DateTimePicker对象事件

请参阅DatePicker的对象事件。

3.3.4.8. TextArea

TextArea是一个提供多行文本输入编辑的组件。



上图是TextArea组件在页面中的呈现，输入框前的文字信息是通过TextArea标签属性prompt来定义的。

3.3.4.8.1. TextArea定义

在screen文件中我们通过<a:textArea>标签来定义一个TextArea对象。

```
<a:textArea name="sql_validation" id="sql_v" bindTarget="sys_parameter_define_ds" width="350">
```

```
    <a:events>
        <a:event handler="login" name="enterdown"/>
    </a:events>
</a:textArea>
```

textArea标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

textArea标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将TextArea对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在TextArea上，另外TextArea上的文字编辑也会立刻修改dataset中的数据。

events标签定义了需要响应的事件函数，例如enterdown事件，当键盘键入回车键时，TextArea会触发enterdown事件，这样我们可以通过配置一个客户端函数login来响应。

```
function login(){
    var lds = $('login_dataset');
    var record = lds.getCurrentRecord();
    Aurora.request({url:'login.svc', para:record.data, success:function(){
        window.location.href='role_select.screen'
    },scope:this});
}
```

3.3.4.8.2. TextArea标签属性

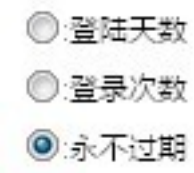
可参阅TextField的标签属性。

3.3.4.8.3. TextArea对象事件

可参阅TextField的对象事件。

3.3.4.9. Radio

Radio是一组单项选择按钮组件。



上图是Radio组件在页面中的呈现。

3.3.4.9.1. Radio定义

在screen文件中我们通过<a:radio>标签来定义一个Radio对象。

```
<a:radio name="state" bindTarget="sys_user_create_ds" layout="vertical"
style="padding-top:5px;padding-bottom:5px;" width="80">
    <a:items>
        <a:item label="SYS_USER.PASSWD_EXPIRED_DAYS" value="1"/>
```

```
        <a:item label="SYS_USER.PASSWD_EXPIRED_TIMES" value="2"/>
        <a:item label="SYS_USER.PASSWD_EXPIRED_NEVER" value="3"/>
    </a:items>
    <a:events>
        <a:event name="change" handler="onRadioChange"/>
    </a:events>
</a:radio>
```

radio标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

radio标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将Radio对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在Radio上，另外Radio上的文字编辑也会立刻修改dataset中的数据。

items标签定义了radio的选项组。items标签下每个item标签即一个选项，label属性指定选项后面的提示性息，value属性指定了选项的值。

events标签定义了需要响应的事件函数，例如change事件，当选中的选项改变为选中另一个选项时，Radio会触发change事件，这样我们可以通过配置一个客户端函数onRadioChange来响应。

```
function onRadioChange(radio, newValue, oldValue){
    var record = $('sys_user_create_ds').getCurrentRecord();
    if(newValue=='1'){
        record.set('password_lifespan_access',null)
        record.getMeta().getField('password_lifespan_days').setReadOnly(false);
        record.getMeta().getField('password_lifespan_access').setReadOnly(true);
    }else if(newValue=='2'){
        record.set('password_lifespan_days',null)
        record.getMeta().getField('password_lifespan_days').setReadOnly(true);
        record.getMeta().getField('password_lifespan_access').setReadOnly(false);
    }else{
        record.set('password_lifespan_access',null)
        record.set('password_lifespan_days',null)
        record.getMeta().getField('password_lifespan_days').setReadOnly(true);
        record.getMeta().getField('password_lifespan_access').setReadOnly(true);
    }
}
```

另外，radio标签还有另一种方式来定义Radio对象，Radio的选项不是用items标签来硬性定义，而是可以用类似于combobox的方法，将选项用options属性绑定到一个dataset数据集，相应的文本提示信息和选项值用labelField和valueField来指定。

```
<a:radio id="roleRadios" labelExpression="${@role_description}_${@company_short_name}"
layout="vertical" options="/model/role_list" valueField="role_company" width="230">
    <a:events>
        <event name="enterdown" handler="goToMain"/>
    </a:events>
</a:radio>
```

labelExpression属性指定如何选项文本提示信息的表达式，可替代labelField属性。

3.3.4.9.2. Radio标签属性

表 3.11.

属性名	用途	默认值	是否必填
labelExpression	指定如何选项文本提示信息的表达式。		
labelField	指定options绑定的选项数据集中用来显示选项文本提示信息的field。	label	
layout	指定按钮组的排列方式。 取值 horizontal | vertical	horizontal	
options	指定Radio选项的数据集。		
valueField	指定options绑定的选项数据集中用来表示选项值的field。		

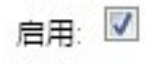
3.3.4.9.3. Radio对象事件

表 3.12.

事件名	用途
blur	失去焦点时触发的事件。
change	选中的选项改变为选中另一个选项时触发的事件。
click	点击选项按钮时触发的事件。
enterdown	敲击回车键时触发的事件。
focus	获得焦点时触发的事件。
keydown	键盘按下时触发的事件。
mouseover	鼠标移到组件上时触发的事件。
mouseout	鼠标移出组件时触发的事件。

3.3.4.10. CheckBox

CheckBox是一个多项选择按钮组件。



上图是CheckBox组件在页面中的呈现，选择框前的文字信息是通过ComboBox标签属性prompt来定义的。

3.3.4.10.1. CheckBox定义

在screen文件中我们通过<a:checkBox>标签来定义一个CheckBox对象。

```
<a:checkBox name="enabled_flag" bindTarget="sys_notify_edit_ds" prompt="FND_OPERATION_UNITS.ENABLED_FLAG">
        <a:events>
        <a:event name="change" handler="onChange"/>
    </a:events>
</a:checkBox>
```

checkBox标签可以设置一个id属性，id是组件的唯一标识，我们可以在页面脚本中用$('id')的方法获得该id对应的组件对象，进而可以调用相应的函数方法。

checkBox标签的bindTarget属性可指定一个dataset对象的id，name属性可指定该dataset其中一个field的名字。这两个属性必须联合使用，其功能是将Radio对象绑定到dataset中的一个field上，进而我们只要对dataset进行操作就能即时反映在Radio上，另外Radio上的文字编辑也会立刻修改dataset中的数据。

events标签定义了需要响应的事件函数，例如change事件，当选中的选项改变为选中另一个选项时，CheckBox会触发change事件，这样我们可以通过配置一个客户端函数onChange来响应。

```
function onChange(checkbox, newValue, oldValue){
    var record = $('sys_user_create_ds').getCurrentRecord();
    if(newValue=='1'){
        record.set('password_lifespan_access',null)
        record.getMeta().getField('password_lifespan_days').setReadOnly(false);
        record.getMeta().getField('password_lifespan_access').setReadOnly(true);
    }else if(newValue=='2'){
        record.set('password_lifespan_days',null)
        record.getMeta().getField('password_lifespan_days').setReadOnly(true);
        record.getMeta().getField('password_lifespan_access').setReadOnly(false);
    }else{
        record.set('password_lifespan_access',null)
        record.set('password_lifespan_days',null)
        record.getMeta().getField('password_lifespan_days').setReadOnly(true);
        record.getMeta().getField('password_lifespan_access').setReadOnly(true);
    }
}
```

3.3.4.10.2. CheckBox标签属性

可参考TextField的标签属性。

3.3.4.10.3. CheckBox对象事件

表 3.13.

事件名	用途
blur	失去焦点时触发的事件。
change	选项的选择状态发生改变时触发的事件。

事件名	用途
click	点击选项按钮时触发的事件。
focus	获得焦点时触发的事件。
mouseover	鼠标移到组件上时触发的事件。
mouseout	鼠标移出组件时触发的事件。

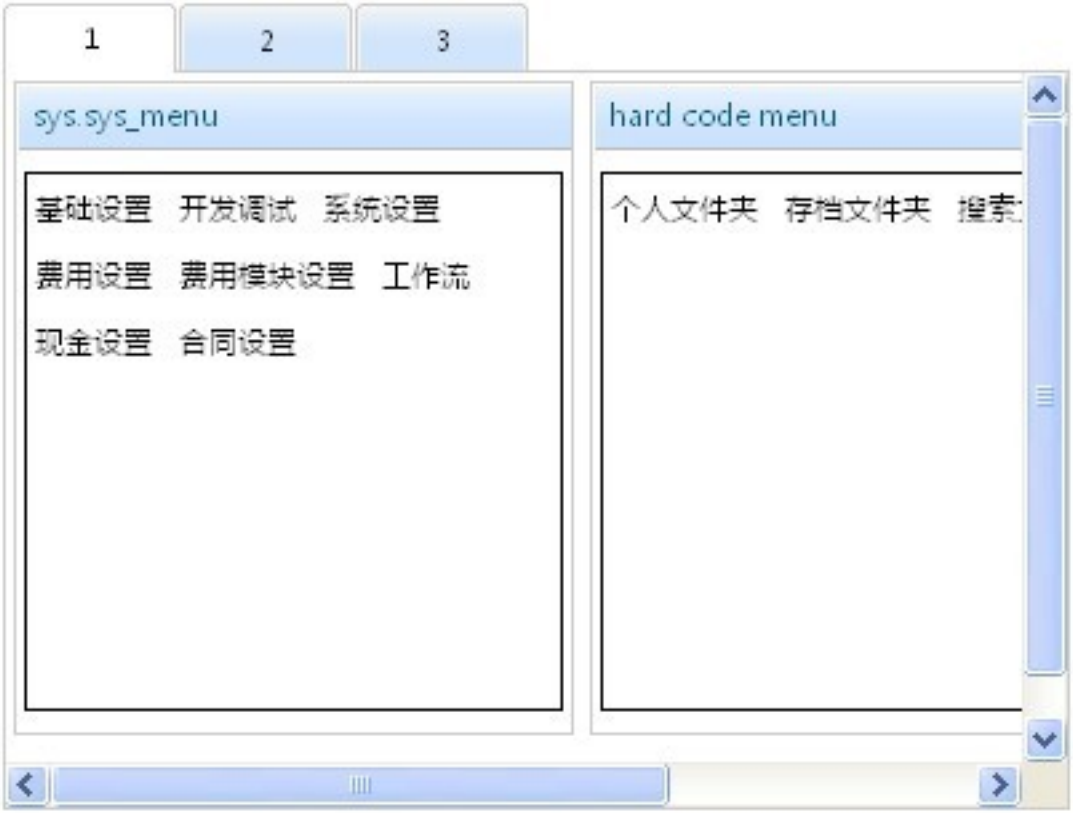
3.3.5. Tab组件

Tab组件是一个标签容器组件，每个标签对应一个页面。

我们通过<a:tabPanel>标签来定义一个tab对象.

```
<a:tabPanel height="300" width="400">
    <a:tabs>
        <a:tab prompt="1" selected="true">
            <a:hBox>
                <a:form id="window" title="sys.sys_menu">
                    <div style="height:200px;width:200px;border:1px solid #000000">
                    </div>
                </a:form>
                <a:form id="window2" title="hard code menu">
                    <div style="height:200px;width:300px;border:1px solid #000000">
                    </div>
                </a:form>
            </a:hBox>
        </a:tab>
        <a:tab prompt="2" ref="datepicker.screen"/>
        <a:tab prompt="3"><span>tab demo!</span></a:tab>
    </a:tabs>
</a:tabPanel>
```

生成的界面如下:



tabPanel标签下定义了tabs标签，tabs标签中每个tab标签即一个标签页。tab标签的属性prompt定义了标签的标题，selected属性定义了标签页是否被选中。

tab标签下可以直接编写screen标签代码，也可以通过属性ref指定引用screen页面的路径，引用页面是延迟加载的，只有当页面被选中时，才会首次加载。

3.3.6. Tree组件

正文 ...

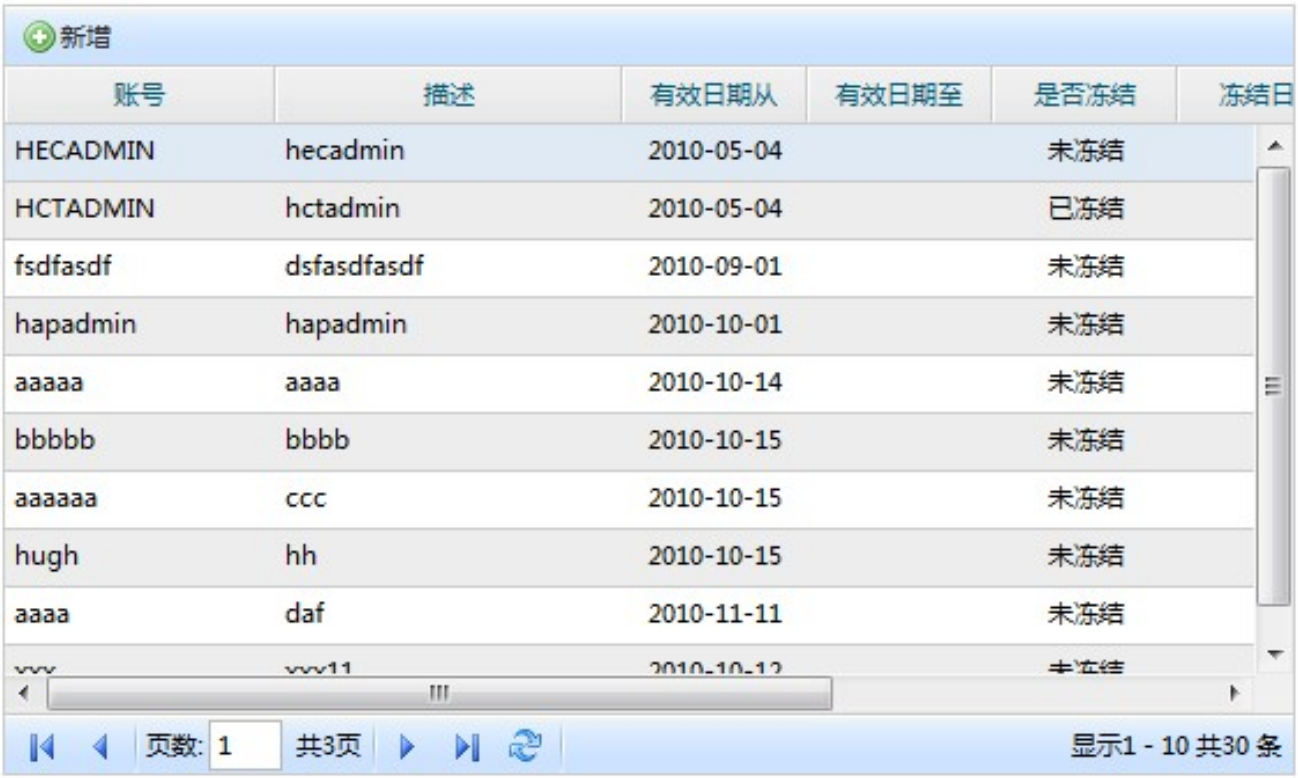
3.3.7. Grid组件

Grid组件是APL中非常重要的一个数据UI组件,它支持锁定,复合表头,排序等特性.

我们通过<a:grid>标签来定义一个grid对象.

```
<a:grid bindTarget="sys_user_result_ds" navBar="true" width="800" id="sys_user_define_grid" height="400">
    <a:toolBar>
        <a:button icon="../../images/add.gif" text="HAP_NEW" click="addUserInfo"/>
    </a:toolBar>
    <a:columns>
        <a:column width="117" name="user_name" sortable="true"/>
        <a:column width="150" name="description" sortable="true"/>
        <a:column width="80" align="center" name="start_date" renderer="Aurora.formatDate" sortable="true"/>
        <a:column width="80" align="center" name="end_date" renderer="Aurora.formatDate" sortable="true"/>
        <a:column width="80" align="center" name="frozen_flag_display" sortable="true"/>
        <a:column width="80" align="center" name="frozen_date" renderer="Aurora.formatDate" sortable="true"/>
        <a:column width="70" name="edit" align="center" prompt="HAP_EDIT" renderer="editUser"/>
    </a:columns>
</a:grid>
```

生成的界面如下:



3.3.7.1. 数据绑定

Grid的数据来源是通过DataSet来获取的,所以Grid需要绑定到一个DataSet上才能正常显示数据.

在grid上我们配置bindTarget属性来指定grid的数据源,bindTarget的值对应到一个具体的dataset的id.

在grid上我们可以配置多个显示列,在列上指定name值,对应到dataset中field的name.

3.3.7.2. 列对齐

column上的align属性,可以指定为center,left,right,对应列中的文字排列方式分别为居中,靠左,靠右.默认值是center

```
<a:grid bindTarget="sys_user_result_ds" navBar="true" width="800" id="sys_user_define_grid" height="400">
    ...
    <a:columns>
        <a:column width="117" name="user_name" align="center"/>
        <a:column width="150" name="description"/>
        ...
    </a:columns>
</a:grid>
```

3.3.7.3. 列锁定

column上的lock属性,可以指定为true或者false,当指定为true的时候当前列将会被锁定.默认值是

false

```
<a:grid bindTarget="sys_user_result_ds" navBar="true" width="800" id="sys_user_define_grid" height="400">
    ...
    <a:columns>
        <a:column width="117" name="user_name" lock="true"/>
        <a:column width="150" name="description"/>
        ...
    </a:columns>
</a:grid>
```

3.3.7.4. 调整列宽

column上的resizable属性,可以指定为true或者false,当指定为true的时候当前列可以动态改变宽度.默认值是true

```
<a:grid bindTarget="sys_user_result_ds" navBar="true" width="800" id="sys_user_define_grid" height="400">
    ...
    <a:columns>
        <a:column width="117" name="money" resizable="false"/>
        ...
    </a:columns>
</a:grid>
```

3.3.7.5. 排序

column上的sortable属性,我们可以指定为true或者false来指定当前列是否可以排序.默认值是false

注意:当点击表头栏的时候会进行排序,这里是通过ajax重新进行了一次查询,是对所有数据的排序.

```
<a:grid bindTarget="sys_user_result_ds" navBar="true" width="800" id="sys_user_define_grid" height="400">
    ...
    <a:columns>
        <a:column width="117" name="user_name" sortable="true"/>
        ...
    </a:columns>
</a:grid>
```

3.3.7.6. 复合表头

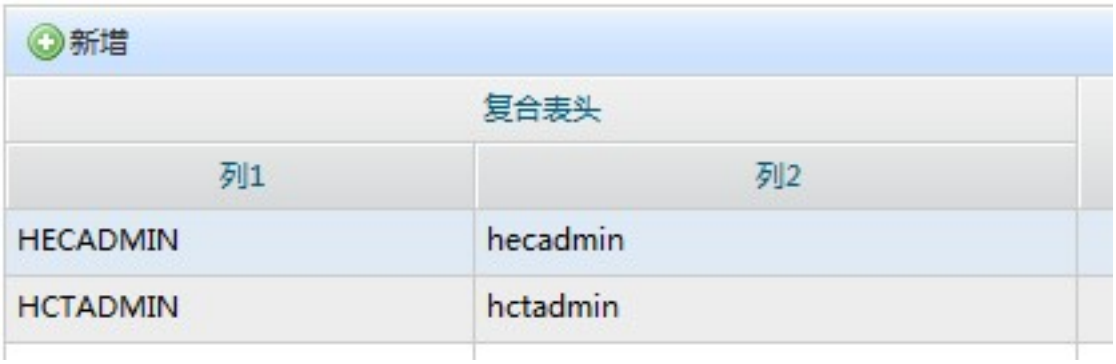
很多情况下我们需要对表头进行组合,这个时候我们可以通过嵌套column列来实现

注意:当点击表头栏的时候会进行排序,这里是通过ajax重新进行了一次查询,是对所有数据的排序.

```
<a:grid bindTarget="sys_user_result_ds" navBar="true" width="800" id="sys_user_define_grid" height="400">
    ...
    <a:columns>
        <a:column prompt="复合表头">
            <a:column width="117" name="user_name" prompt="列1"/>
```

```
            <a:column width="117" name="user_name" prompt="列2"/>
            ...
        </a:column>
        ...
    </a:columns>
</a:grid>
```

显示效果:



3.3.7.7. 列渲染

渲染函数(renderer)

很多情况下我们并不需要直接显示数据,而是需要对数据进行一下加工,例如对于大于0的金额显示绿色,小于0的金额显示红色.这个时候我们就需要 在column上指定renderer渲染函数来实现.

首先我们需要实现一个renderer函数,来判断当前金额,然后返回一段html代码.

```
function moneyRenderer(value,record, name){
    if(value >=0){
       return '<font color="green">'+value+'</font>'
    }else{
        return '<font color="red">'+value+'</font>'
    }
}
```

接下来在column上指定renderer属性为moneyRenderer.

```
<a:grid bindTarget="sys_user_result_ds" navBar="true" width="800" id="sys_user_define_grid" height="400">
    ...
    <a:columns>
        <a:column width="117" name="money" renderer="moneyRenderer"/>
        ...
    </a:columns>
</a:grid>
```

3.3.7.8. 汇总列

grid提供了汇总的功能,我们可以通过在列上配置footerRenderer来实现.

例如:我们需要对金额列进行汇总.首先需要实现一个汇总的函数summaryRenderer

```
function summaryRenderer(datas,name){
    var sum = 0;
    for(var i=0;i<datas.length;i++){
        var r = datas[i];
        var d = r.get(name);
        var n = parseFloat(d);
        if(!isNaN(n)){
            sum +=n;
        }
    }
    return '合计金额: <font color="red">' + Aurora.formatNumber(sum) + '</font>';
}
```

接下来在column上指定footerRenderer属性为summaryRenderer.

```
<a:grid bindTarget="sys_user_result_ds" navBar="true" width="800" id="sys_user_define_grid" height="400">
    ...
    <a:columns>
        <a:column width="117" name="money" footerRenderer="summaryRenderer"/>
        ...
    </a:columns>
</a:grid>
```

注意:这里的summaryRenderer的数据范围只能局限在当前页.无法统计到所有数据.换句话说无法统计到下一页的数据

3.3.7.9. 工具栏

在Grid组件上提供了一个工具栏,Grid本身提供了内置的几个按钮,我们还可以自定义新的按钮.

```
<a:grid id="grid" bindTarget="sys_function_result_ds" height="300" navBar="true" width="500">
    <a:toolBar>
        <a:button type="add"/>
        <a:button type="delete"/>
        <a:button type="save"/>
    </a:toolBar>
    <a:columns>
        ...
    </a:columns>
</a:grid>
```

我们可以看到在grid上我们添加了3个按钮.通过指定type值可以确定这个按钮的逻辑.

Grid组件默认提供了3种类型的按钮.

- add -- 在当前grid中新增一条记录.
- delete -- 删除当前选中的记录.

- save -- 保存修改过的记录.

除了Grid内置的按钮,我们还可以自定义按钮

```
<a:grid id="grid" bindTarget="sys_function_result_ds" height="300" width="500">
    <a:toolBar>
        <a:button icon="../../images/add.gif" text="HAP_NEW" click="addUserInfo"/>
    </a:toolBar>
    <a:columns>
        ...
    </a:columns>
</a:grid>
```

icon代表按钮的图标,text表示按钮的文本内容,click指定当前按钮的动作.

3.3.7.10. 编辑器

大部分情况下我们需要对数据进行编辑,APL的Grid组件可以集成编辑器,只需要在列中配置editor.

```
<a:grid id="grid" bindTarget="sys_function_result_ds" height="300" width="500">
    <a:toolBar>
        <a:button type="add"/>
        <a:button type="delete"/>
        <a:button type="save"/>
    </a:toolBar>
    <a:columns>
        <a:column name="function_code" editor="sys_function_result_grid_tf" width="100"/>
        <a:column name="function_name" editor="sys_function_result_grid_tf" width="120"/>
        <a:column name="parent_function_name" editor="sys_function_result_grid_lv" width="120"/>
        ...
    </a:columns>
    <a:editors>
        <a:textField id="sys_function_result_grid_tf"/>
        <a:lov id="sys_function_result_grid_lv"/>
    </a:editors>
</a:grid>
```

这里我们可以看到Grid中增加了一个editors子标签,在editors标签下定义了几个编辑器.我们可以把ediotrs理解为grid下的一个组件库. 每一个编辑器需要指定一个唯一的id,然后在列上配置editor指定你所需要的编辑器id.这样当你点击grid的某一个单元格的时候,grid 首先回去找列中对应的editor值,然后再去组件中根据id来查找,最后显示到指定的单元格上.

3.3.8. Table组件

Table组件是一个数据UI组件，类似于Grid组件。

我们通过<a:table>标签来定义一个table对象.

```
<a:table id="sys_user_define_grid" bindTarget="sys_user_result_ds" percentWidth="90"
        style="margin:7px;" title="SYS_USER.USER_SEARCH">
    <a:columns>
```

```
        <a:column>
            <a:column name="user_name" footerRenderer="frdr" percentWidth="10"/>
            <a:column name="description" editor="description_tf" percentWidth="20"/>
        </a:column>
        <a:column name="start_date" align="center" footerRenderer="frdr"
                percentWidth="10" renderer="Aurora.formatDate"/>
        <a:column name="end_date" align="center" percentWidth="10" renderer="Aurora.formatDate"/>
        <a:column name="frozen_flag_display" align="center" percentWidth="10"/>
        <a:column name="frozen_date" align="center" editor="frozen_date_table_dp"
                percentWidth="10" renderer="Aurora.formatDate"/>
        <a:column name="assign_role" align="center" percentWidth="10" prompt="SYS_USER.ROLE_ASSIGN"
                renderer="assignRole"/>
        <a:column name="set_password" align="center" percentWidth="10" prompt="MODIFY_PASSWORD"
                renderer="setPassword"/>
        <a:column name="edit" align="center" percentWidth="10" prompt="HAP_EDIT" renderer="editUser"/>
    </a:columns>
    <a:editors>
        <a:datePicker id="frozen_date_table_dp"/>
        <a:textField id="description_tf"/>
    </a:editors>
</a:table>
```

生成的界面如下:

用户查询

账号	描述	有效日期从	有效日期至	是否冻结	冻结日期	分配角色	修改密码	编辑
HECADMIN	hecadmin	2010-05-04		未冻结		分配角色	修改密码	编辑
HCTADMIN	hctadmin	2010-05-04		已冻结		分配角色	修改密码	编辑
fsdfasdf	dsfasdfasdf	2010-09-01		未冻结		分配角色	修改密码	编辑
hapadmin	hapadmin	2010-10-01		未冻结		分配角色	修改密码	编辑
aaaaa	aaaa	2010-10-14		未冻结		分配角色	修改密码	编辑
bbbbb	bbbb	2010-10-15		未冻结		分配角色	修改密码	编辑
aaaaaa	ccc	2010-10-15		未冻结		分配角色	修改密码	编辑
hugh	hh	2010-10-15		未冻结		分配角色	修改密码	编辑
aaaa	daf	2010-11-11		未冻结		分配角色	修改密码	编辑
ab	ab	2011-02-01		未冻结		分配角色	修改密码	编辑
xxx	xxx11	2010-10-12		未冻结		分配角色	修改密码	编辑
xxxx	xxxx	2010-10-11		未冻结		分配角色	修改密码	编辑
w334324	edsfasd	2010-09-01		未冻结		分配角色	修改密码	编辑
ccc	ccc	2010-09-03		未冻结		分配角色	修改密码	编辑
23	221	2011-02-24	2011-02-26	未冻结		分配角色	修改密码	编辑
1		1						

3.3.8.1. Table与Grid的异同

- Table没有高度height属性，其高度是自适应高度，随着行数的增加而增高。
- Table和Table有绝对宽度width属性的同时，还有百分比宽度percentWidth属性。
- Table有title属性，可为table增加一个标题。
- Table没有列锁定功能。

- Table没有调整列宽功能。
- Table没有排序功能。
- Table没有工具栏toolBar。
- Table没有导航栏navBar。
- Table没有分页，所有数据将成列在一页中。

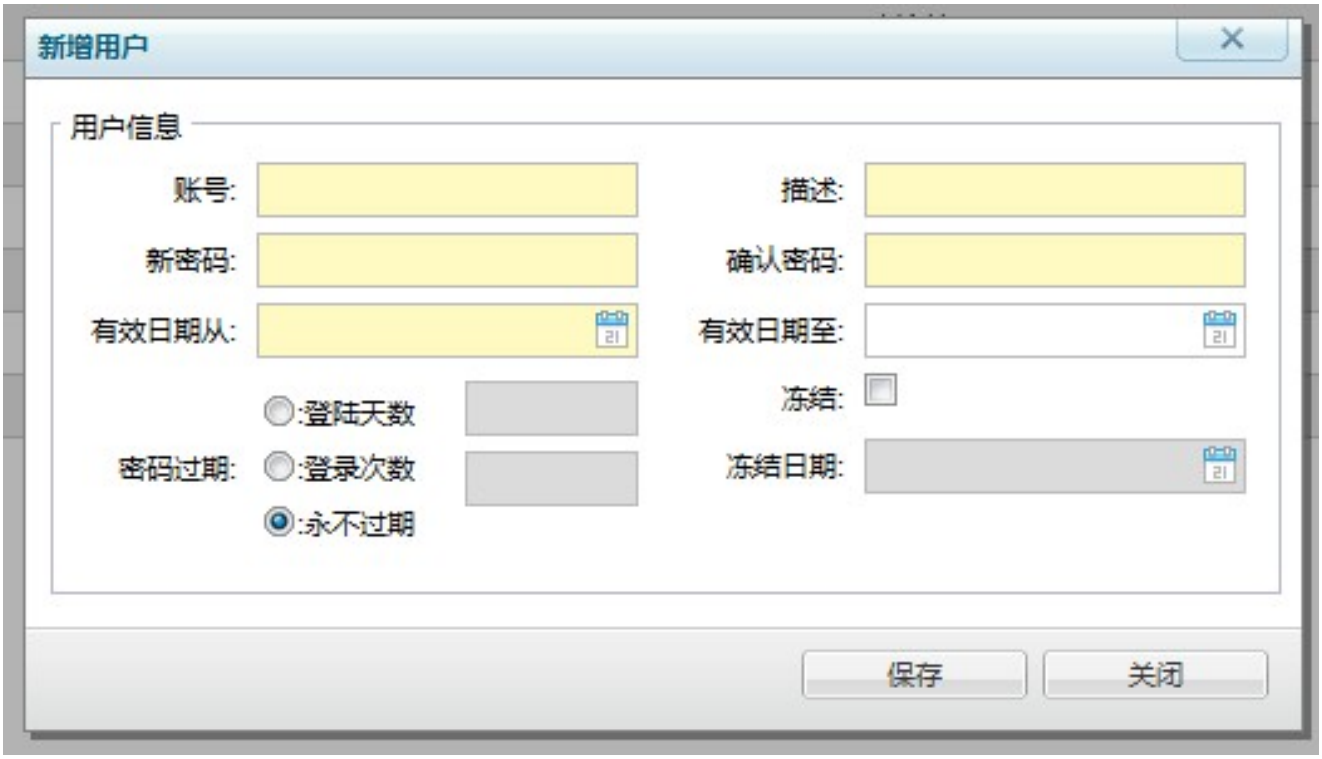
3.3.9. 窗口组件(Window)

APL中提供的窗口组件不同于html中的原生window,它是通过div模拟出来的.本质上和父页面是在同一个dom树中.

window组件的内容是通过ajax动态加载进来的.所以加载页面中的所有对象在父页面中都可以直接获取到,反过来window也可以直接获取父页面的所有对象.

window组件的创建都是通过JavaScript脚本创建的.

```
function openWindow(){
    var win = new Aurora.Window({id:'mywin', url:'user.screen',title:'窗口', height:400,width:700});
}
```



通过new Aurora.Window我们可以创建出一个window窗口.id是窗口的唯一标识,url指定当前窗口需要加载的screen文件.title指定打开窗口的标题,height和width分别指定窗口的大小.

表 3.14.

参数名	用途	默认值	是否必填
id	window窗口的id。		true
title	window窗口的标题。		false
url	window窗口的url。		true
height	window窗口的高度。	400	false
width	window窗口的宽度。	350	false

有时候我们仅仅需要显示一个提示信息,或者一个简单的警告.这个时候通过url加载就比较麻烦. APL为我们提供了几个通用的窗口函数.

- 提示信息窗口

```
Aurora.showMessage(title, msg,callback,width,height);
```

表 3.15.

参数名	用途	默认值	是否必填
title	window窗口的标题。		true
msg	需要提示的信息内容		true
callback	回调函数,不需要的话用null		false
height	window窗口的高度。	100	false
width	window窗口的宽度。	300	false

- 带警告图标的窗口

```
Aurora.showWarningMessage(title, msg,callback,width,height);
```

配置参数参考"提示信息窗口"

- 带信息图标的窗口

```
Aurora.showInfoMessage(title, msg,callback,width,height);
```

配置参数参考"提示信息窗口"

- 带错误图标的窗口

```
Aurora.showErrorMessage(title, msg,callback,width,height);
```

配置参数参考"提示信息窗口"

- 带确定取消按钮的确认窗口

```
Aurora.showComfirm(title, msg, okfun,cancelfun, width, height);
```

表 3.16.

参数名	用途	默认值	是否必填
title	window窗口的标题。		true
msg	需要提示的信息内容		true
okfun	确定按钮回调函数,不需要的话用null		
cancelfun	取消按钮回调函数,不需要的话用null		
height	window窗口的高度。	100	false
width	window窗口的宽度。	300	false

3.3.10. 上传组件

正文 ...

3.3.11. TreeGrid

TreeGrid是一个树状结构的表格组件,它集成了Grid和Tree的多种特性.

```
<a:treeGrid bindTarget="function_tree_ds" expandField="_expanded" height="400" id="functionTreeGrid"
    idField="function_id" parentField="parent_function_id" showCheckBox="true" width="570">
    <a:columns>
        <a:column name="function_name" prompt="功能名称" width="250"/>
        <a:column name="function_code" prompt="功能代码" width="120"/>
        <a:column editorFunction="expandEditorFunction" name="expanded" prompt="是否展开" width="80"/>
        <a:column align="right" editor="grid_nf" name="sequence" prompt="序列号" width="100"/>
    </a:columns>
    <a:editors>
        <a:numberField id="grid_nf"/>
        <a:checkBox id="grid_cb"/>
    </a:editors>
</a:treeGrid>
```

表 3.17.

属性名	用途	默认值	是否必填
bindTarget	组件所绑定的dataset数据集，属性值是dataset的ID。		
className	组件的样式表。		
emptyText	当组件没有值的时候显示在组件上的提示信息。		
id	组件的唯一标识，可用$(id)方法获得组件对象。		
marginWidth	组件与窗口之间的宽度差，单位像素(px)，可以根据窗口宽度的改变而改变。		
name	组件对应dataset数据集中的一个字段field，属性值是字段名。		
prompt	输入框前的提示文字，默认调用BM的prompt。		
readOnly	设定组件是否只读。 取值 true | false	false	
required	设定组件是否必填。 取值 true | false	false	
style	组件的样式。		
typeCase	组件的大小写输入限制。		

属性名	用途	默认值	是否必填
	取值 upper | lower		
width	组件的宽度，单位像素(px)。	150	

3.4. 个性化及定制

个性化及定制：

3.4.1. 界面定制

界面定制及更改：

3.4.2. 组件样式修改

组件样式修改：

3.4.3. 修改网页整体布局

修改网页整体布局：

3.5. 多语言支持

个性化及定制：

3.5.1. 基于数据库存储的多语言支持

基于数据库存储的多语言支持：

3.5.2. 自定义多语言实现

自定义多语言实现：

3.5.3. Screen及模板资源文件中的多语言支持

xxx：

第 4 章 Aurora服务层(ASL)

4.1. ASL(Aurora Service Layer)总体架构概述

ASL结构图：

4.2. 业务模型 (Business Model)

业务模型

4.2.1. BusinessModel概述

正文...

4.2.2. 基础配置：表、字段、主键、筛选条件

正文...